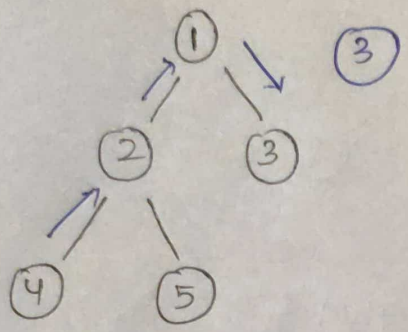
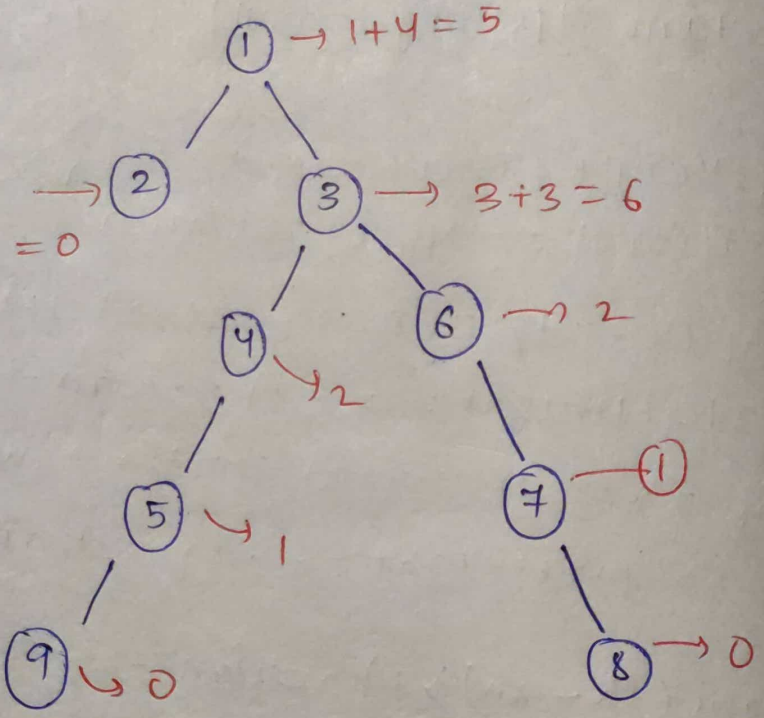
Diameter of a Binary Tree :-

→ Longest path b/w 2 nodes.

→ path does not need to pass via root.



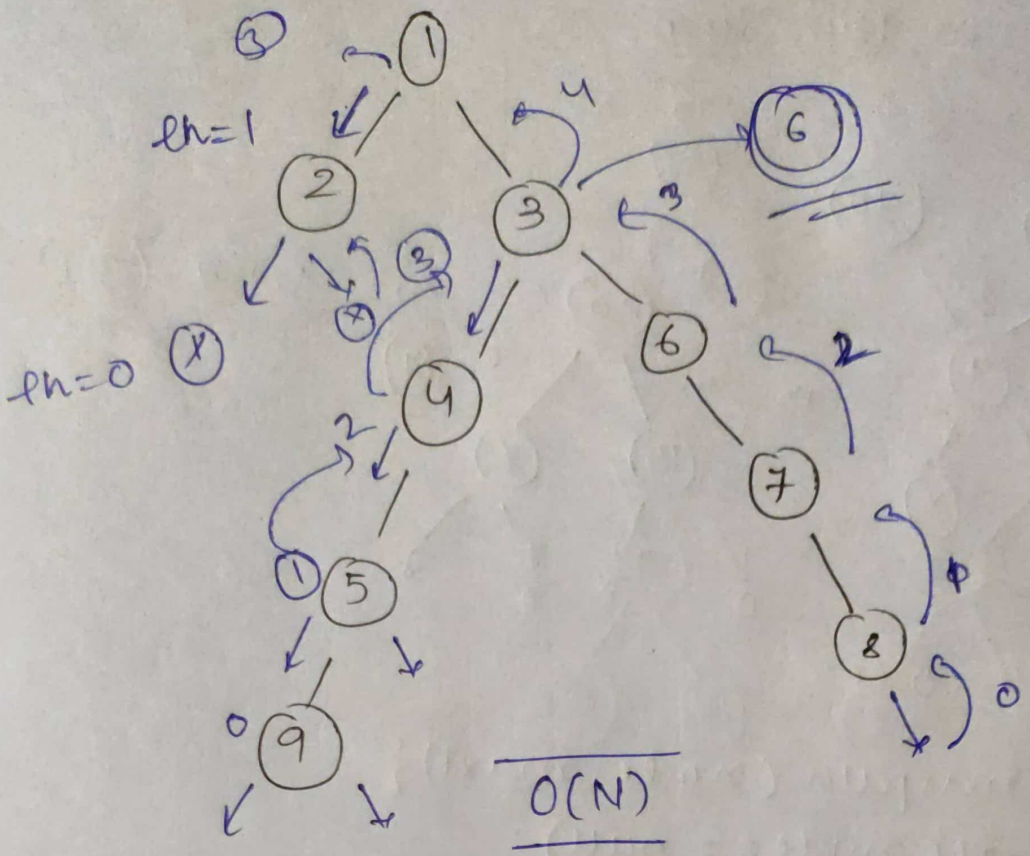


For every node calculate its lh+rh

Brute Force :-

```
find Max (Node) {
    if (root == NULL)
        return;
    lh = findleftH (node->left);
    rh = findrightH (node->right);
    maxi = max (maxi, lh+rh);
    findMax( node->left);
    findMax (node -> right);
}
```

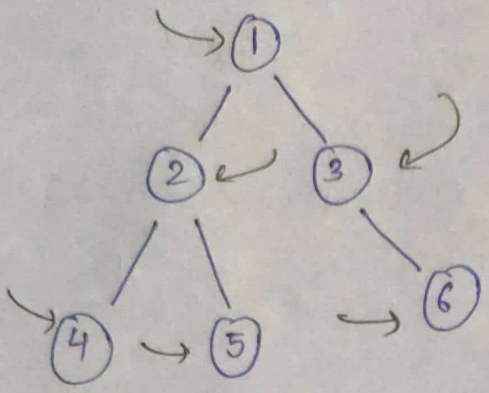
O(N²)

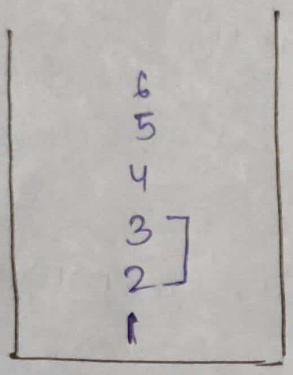


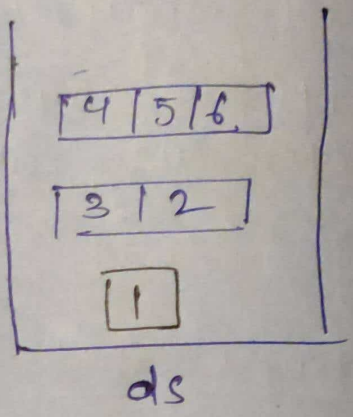

```
int diameterOfBinaryTree(TreeNode *root) {
    int diameter = 0;
    height(root, diameter);
    return diameter;
}

int height(TreeNode *node, int &diameter) {
    if(!node) return 0;
    int lh = height(node->left, diameter);
    int rh = height(node->right, diameter);
    diameter = max(diameter, lh+rh);
    return 1 + max(lh, rh);
}
```

ZigZag Traversal of a Binary Tree







Code:-

```
vector<vector<int>> zigzag (TreeNode *root){
    vector<vector<int>> result;
    if(root==NULL){
        return res;
    }
queue<TreeNode*> nodeQueue;
nodeQueue.push(root);
bool lefttoright = true;
    while(!nodeQueue.empty()){
        int size= nodeQueue.size();
        vector<int> row(size);
    for(int i=0; i< size ;i++){
        Treenode * node = nodeQueue.front();
        nodeQueue.pop();
```

```
int index = leftToRight ? i : size-1-i;
row[index] = node->val;
if (node->left) {
    nodeQueue.push(node->left);
}
if (node->right) {
    nodeQueue.push(node->right);
}
}
// after this level.
leftToRight = !leftToRight;
result.push_back(row);
}
return result;
}
```